ECE 354 – Computer Systems Lab II
Compilation for PIC and DSPs
Exam

• Generally, it went well
  — Max 92, avg 72, min 41
• Grades are relative
• Solutions
  — Available in lab
• If you have any questions about the grading, see me.
Lab 5

- Lab 5 is an INDIVIDUAL PROJECT
- Lab 5 has two parts:
  - Compilation to PIC
  - Compilation to DSP
- All software lab, no hardware required
  - Return hardware Wednesday 5/5/04 2:30 in lab
- Demos due 5/13 and 5/14/04
  - Will keep 30 minute team sign-up slots
- Report due 5/17/04
  - Will be shorter report than for other labs
Compiling for the PIC

- Compilers convert “high-level” language to assembly code
  - Example: C/C++ compiler, Pascal compiler, ...
- Assembly code is then assembled to binary code
  - Same as in lab 0-4
- What are the benefits / drawbacks for compiling?
Compilers

- **Benefits:**
  - Easier to write programs
  - More likely to produce correct code
  - Can hide implementation details from user (e.g., memory banking)
  - Can do code optimization

- **Drawbacks:**
  - Possibly less efficient code
  - Difficult to match PIC components to programming abstractions
    - Timers, A/D, UART, etc.
  - Possibly cannot use all hardware component efficiently
  - Harder to debug
We use free C-compiler:

- [http://www.bknd.com/cc5x/downl-stud.shtml](http://www.bknd.com/cc5x/downl-stud.shtml)
- Limited functionality, for academic use only
Lab 5 – Part I

- Compare performance of compiled code vs. your assembly skills
  - Application: bubble-sort
- Measure time it takes to execute code
  - Compare compiled code vs. hand-coded assembly
- Time measurement on PIC:
  - Window -> Stopwatch
- Data is put in “by hand” in register file window
Bubble Sort (1)

- Check neighboring pairs of elements for correct order
- If not ordered, swap
- Repeat until sorted
- Sequence of comparisons is important
- Largest values “bubble” to the top
  - Don’t need to be compared again
Bubble Sort (3)
Details

- Enter data to be sorted in registers “by hand”
  - Window -> Fill Registers
  - Right-click on data

- Run on several different data sets
  - Different data will change results
  - Use worst-case and best-case as extremes

- Note results from different runs for lab report
  - Write down data and run-time
Lab 5 – Part II

• Digital Signal Processors (DSP)
  — Another type of embedded processor
  — Instead of controlling peripherals (PIC), a DSP processes signals

• Examples of DSP applications:
  — CD players
  — Mobile phones
  — Hard drive controllers
  — Communication equipment
  — Speech processing

• Why bother?
  — You should see another type of architecture in ECE 354
  — DSPs provide significantly more performance for certain apps
  — You might use a DSP in your senior design project
Example DSP Application

- Digital filtering of signals
  - Requires A/D and D/A converters
- Programmable for different purposes
- More precise than analog circuitry
  - Not temperature dependent
  - More accurate for low frequencies

A/D Converter → sampled digitized signal → DSP → digitally filtered signal → D/A Converter → filtered analog signal
DSP Filters (1)

- Notation conventions
  - \( x_i \) are data points that are sampled by A/D converter
  - \( y_i \) are output values generated by DSP
  - \( n \) is current time
  - \( a \) are filter coefficients

- Example filters:
  - Simple gain filter: \( y_n = K \cdot x_n \)
    - Depends only on current input value
  - Two-term average filter: \( y_n = \frac{1}{2} \cdot (x_n + x_{n-1}) \)
    - Depends on past input values
  - No dependency on output (Finite Impulse Response (FIR))
Infinite Impulse Response (IIR) filter examples:
  - Dependency on past output: \( y_n = x_n + y_{n-1} \)

“Order” of a filter:
  - Uses \( n \) past values, then \( n^{th} \) order filter
  - Example: \( y_n = \frac{1}{2} \times (x_n + x_{n-1}) \) is first order FIR filter

General specification:
  - FIR: \( y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} + \ldots \)
  - IIR: \( y_n = \frac{(a_0x_n + a_1x_{n-1} - b_1y_{n-1})}{b_0} \)
    or \( b_0y_n + b_1y_{n-1} = a_0x_n + a_1x_{n-1} \)

“Multiply and accumulate” is fundamental operation of DSP
Multiply and Accumulate

- Basic architecture:
  - Requires delay element
  - Multipliers implement coefficients
- Example is second order FIR filter
  - Why?
- IIR filter requires feedback
DSP Characteristics

- Typically for applications with high bandwidth and high computational requirements
- A/D and D/A converters
- Typically Harvard architecture
- Support floating or fixed point operations and multiplications
- Multiple ALU operations in same clock cycle
  - More parallelism gives more performance
- Multiple simultaneous memory accesses
- Structured to do multiply/accumulate operations
  - Necessary for filter operations
DSP Architecture

- Texas Instruments TMS3200C6201
- Similarities to PIC:
  - Harvard Arch.
  - Timers
  - Interrupts
  - ICD
- Differences to PIC:
  - Multiple data paths
  - External memory interface
  - ...

![Diagram of DSP Architecture](attachment:image.png)
DSP Data Path

- L1, S1: arithmetic, logic, branch instructions
- M1: multiplier
- D1: data transfer between register file and memory and transfer to other data path
- Same units on second data path
- All units can operate in parallel (VLIW)
Programming DSPs

- VLIW (Very Large Instruction Word)
  - One “instruction” controls all eight data path units
- Hard to program by hand
  - User programs in C
  - Compiler figures out dependencies and some optimizations
  - User can optimize code by hand
- For real-time applications:
  - Repeated optimization, profiling, more optimizations, ...
- Lab 5: compare performance of code with different levels of optimization
  - Application: dot product on vectors (multiply and accumulate)
Lab 5 – DSP Part

- You are given C code that computes vector sum
- You need to add code that computes dot product
- Compile code and simulate
  - Program reports number of cycles used as output
  - Compile with different levels of optimization
- Compare DSP assembly code
  - Understand reasons for speedup
  - See how many parallel instructions can be issued
- What you will learn
  - More parallelism results in faster code
  - Optimization is important part of DSP programming
Lab 5 – Assembly Comparison

lab5.o1.asm - Notepad

```assembly
; **

L9:
LDH .D2T2 "*+B4(2),B5" ; |92|
ADD .L1 1,A0,A0 ; |91|
LDH .D1T1 "*+A3[A0],A5" ; |92|

B1]
CMPLTU .L2X A0,B0,B1 ; |91|
B .S1 L9 ; |91|
NOP 2 ; |91|
MPY .M1X B5,A5,A5 ; |92|
NOP 1 ; |91|
ADD .L1 A5,A4,A4 ; |92|
; BRANCH OCCURS ; |91|
```

lab5.o2.asm - Notepad

```assembly
; **

L18: ; PIPED LOOP KERNEL

ADD .L1 A6,A5,A5 ; |92|
MPY .M1 A4,A0,A6 ; |92|
B .S2 L18 ; |92|
SUB .L2 B0,1,B0 ; |91|
LDH .D1T1 "*A3++,A0" ; |92|
```

Lab 5 Summary

- Part I: Bubble sort – C compiler vs. you
  - Write bubble sort in C and compile to assembly
  - Write bubble sort in assembly
  - Compare run-time performance for both
  - Perform 5 different measurements (data sets) for each
  - Show comparison

- Part II: DSP code optimization
  - Use code skeleton and at dot product for vectors
  - Compile with three different optimization levels
  - Simulate code and compare number of cycles spend
  - Look at generated assembly code and understand why
  - Optional: improve your code and reduce cycles

- Reminder: individual project – everyone has to do it…
Challenge

For both parts of the lab, a competition is held. For both competitions, the goal is to write the fastest code:

1. PIC code: write the **fastest bubble sort** that can sort the worst case data array.
2. DSP code: write the **fastest dot product** function.

The winner of each competition receives an **extra 10 points** for lab 5 (the total of lab 5 cannot exceed 100 points). If there are multiple competitors who achieve the best result, the 10 points will be divided between them (fractions of points are rounded off). If there is a single winner for the PIC competition, he or she may carry the title **“ECE354 Master Assembly Crafter.”** If there is a single winner for the DSP competition, he or she may carry the title **“ECE354 Master Parallelism Exploiter.”** If both competitions are won by the same individual, he or she may carry the title **“ECE354 Supreme Embedded Coder.”** During lab hours, the current best performance should be announced on the blackboard in the lab. All competitors must have their solution’s performance announced on the blackboard by Wednesday 5/12/04 to give others a chance to counter (updates due to code improvements are allowed and encouraged).
References

• Part I:
  — On course web page: link to C compiler, example C code
  — Bubble sort: read today’s slides or look on web

• Part II:
  — DSP Data Sheet and Programmer's Guide on course web page
    (recommended: Programmer’s Guide Chapter 2)
Miscellaneous

• Return of lab bench hardware and keys
  — Wednesday 2:30 in lab
    • One member of each group must be there
  — If you have broken components, please tell Keith – don’t just put them back

• Course Evaluation
  — In just one minute…
The End...

• ...almost
• I hope you enjoyed the course
• Hopefully you have learned about:
  – Programming PICs
  – Integrating PICs with other hardware
  – General system concepts: interrupts, timers, A/D conversion, buses, serial communication, etc.
  – Debugging your software and hardware
  – Teamwork
• There’s a good chance you’ll be able to apply your knowledge in the real world
  – SDP next semester
  – At your job